Pergamon

Neural Networks, Vol. 10, No. 6, pp. 1133–1141, 1997

Printed in Great Britain
0893–6080/97 $17.00+.00

PII: S0893-6080(97)00032-4

CONTRIBUTED ARTICLE

Network Synthesis through Data-Driven Growth and Decay

CHUANYI JI[1] AND DEMETRI PSALTIS[2]

[1]Department of Electronics, Computing and Systems Engineering, Rensselaer Polytechnic Institute and
[2]Department of Electrical Engineering, California Institute of Technology

(*Received* 1 *August* 1996; *accepted* 21 *January* 1997)

Abstract—*An algorithm for addition and deletion (ADDEL) of resources during learning is developed to achieve two goals: (1) to find feed-forward multilayer networks that are as small as possible, (2) to find an appropriate structure for such small networks. These goals are accomplished by operating alternately between an adding phase and a deleting phase while learning the given input-output associations. The adding phase develops a crude structure by filling in resources (connections, units and layers) at a virtual multilayer network with a maximum of L possible layers. The deleting phase then removes any unnecessary connections to obtain a refined structure. The additions and deletions are done based on a sensitivity measure and a corresponding probability rule so that only the synapses which are most effective in reducing the output error are preserved. A generalization error estimated from a validation set is used to control the alternation between the two learning phases and the termination of learning. Simulations, including handwritten digit recognition, demonstrate that the algorithm is effective in finding an appropriate network structure for a small network which can generalize well. The algorithm is used to investigate when the size of a network is important for generalization.* © *1997 Elsevier Science Ltd.*

1. INTRODUCTION

In this paper we describe a learning algorithm which develops from data the structure of the network and the values of the connections. Several learning algorithms of this type have been previously developed. They can be roughly divided into two categories: structure minimization through weight decay and deletion, and structure growth by adding resources. Most algorithms that use weight deletions solve a regularization problem. That is, a network with a fixed structure is used and an extra term is added as a regularizer to the original energy function in a setting of supervised learning. Such a term usually measures the complexity of the network (Denker, Le Cun & Solla, 1989; Nowlan & Hinton, 1992; Weigend, Rumelhart & Huberman, 1990; Ji, Snapp & Psaltis, 1990) and constrains the network to learn the training samples using as few degrees of freedom as possible. On the one hand, although these methods have been shown to be capable of finding networks which generalize well, they require either a very slow tuning of weights from a fixed network architecture or a lot of prior knowledge in the network design. On the other hand, learning algorithms for network structure growth (Fahlman & Lebiere, 1989; Frean, 1990; Nadel, 1989) add neurons and layers during the learning phase without requiring a fixed structure to be prespecified. However, since these algorithms usually control additions of resources by trying to learn the training set as well as possible, overfitting rather than generalization may occur. The learning algorithm (ADDEL) presented in this paper combines both structure minimization and growth, and tries to circumvent their individual drawbacks. Specifically, the algorithm has three salient features.

First, it tries to find as small a network as possible that learns the training samples.

Second, the architecture of the network is determined through two learning phases: an adding phase and a deleting phase, while the network learns the given input–output associations using gradient descent. The adding phase builds up a crude structure by filling the connections and neurons at a large "virtual" multilayer network with a maximum of L possible layers. The deleting phase refines the structure by trimming off any unnecessary connections. The adding and deleting are

Acknowledgements: The authors thank Dr Alan Yamamura for the collaborations on the simulations on two-link manipulator. The support of NSF and ARPA is gratefully acknowledged.

Requests for reprints should be sent to Chuanyi Ji, Department of Electonics, Computing and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, U.S.A.

controlled by sensitivity measures so that only those connections which are most effective in reducing the error are included. This method allows us to search for an appropriate network structure without specifying a fixed structure a priori.

Third, an estimated generalization error evaluated on line through a validation set is used to control the alternation between the two learning phases and termination of learning.

Theoretical investigations lead to the conclusion that small networks should generalize better than large ones (Barron, 1991; Baum & Haussler, 1989). In practice, however, since it is not always true that each weight can be considered as one degree-of-freedom, under-trained large networks can also generalize well on some occasions (Martin & Pittman, 1991). We use our algorithm to investigate experimentally whether, and when, a small network is needed to obtain good generalization. Our investigations on learning different mappings, including both analog and binary mappings, show that small networks are necessary for good generalization when the mapping is "difficult". When a mapping is "easy", both under-trained large networks and small networks can generalize well. The specific "difficult" and "easy" mappings considered in this paper will be explained later, and more rigorous specifications will be given in subsequent research.

The paper is organized as follows. Section 2 explains the algorithm and Section 3 gives simulation results. Section 4 will address the problem on small and under-trained large networks by providing comparisons between the performance of under-trained large networks and small networks. The conclusion will summarize the results and state some related issues.

2. LEARNING ALGORITHM

The algorithm develops network architecture through two phases: an adding phase and a deleting phase.

Let L be the chosen maximum number of layers possible for a virtual feedforward multilayer networks with $n(l)$ units at layer l, where $0 \le l \le L$ ($l = 0$ refers to the input layer). Here $n(0)$ and $n(L)$ are fixed and the remaining $n(l)$ are adaptable. The transfer function of all the neurons is the sigmoid function $f(x) = tanh(x)$, and the quadratic energy function is used for training:

$$E = \sum_i \|\vec{y}_i - \vec{t}_i\|^2.$$

The $\vec{y}_i$s and $\vec{t}_i$s denote the ith actual and desired outputs of the network, respectively. The modification of the weights is done using the gradient descent method.

2.1. Adding Phase

The adding process includes adding new units as well as connections at different layers. The criterion for addition is based on a sensitivity measure on connections which in turn provides a sensitivity measure for units at different layers.

Specifically, suppose $n(l)$ units already exist at the l-th layer of the virtual multilayer network, where $0 \le l \le L$ (the 0-th layer is the input layer). When we consider the $n(l) + 1$-th candidate unit at layer l for possible addition, we first randomly choose small outgoing connections for the candidate unit and then use a sensitivity criterion to choose its incoming connection $w(i,n(l) + 1,l)$, where $1 \le i \le n(l-1)$ and $1 \le l \le L$. Here $w(i,j,l)$ denotes the connection from unit i at the $(l-1)$-th layer to unit j at the l-th layer for $1 \le i \le n(l-1)$, $1 \le j \le n(l)$, and $1 \le l \le L-1$. Specifically, we have the sensitivity of $w(i,(n(l) + 1),l)$ defined as

$$S_{i(n(l)+1)l} = \left[\frac{\partial E}{\partial w(i, n(l)+1, l)}\right]^2. \tag{1}$$

This sensitivity measures how effective a connection associated with a candidate unit can be in reducing the training error, if added. The sensitivity S_l of the candidate unit $n(l) + 1$ at layer l is just the summation of the sensitivities over all its incoming connections. That is,

$$S_l = \sum_{i=1}^{n(l-1)} S_{i(n(l)+1)l}. \tag{2}$$

Similarly, S_l ($1 \le l \le L-1$) indicates how effective a candidate unit can be, if added, in reducing the error. Since S_l depends on the outgoing weights of the candidate units which are chosen randomly, S_l is also a random varible. Therefore, a probabilistic rule is needed to choose a candidate unit. The probability that the candidate unit at layer l gets selected is defined as

$$\Pr(\text{a unit at layer } l \text{ is chosen}) = \frac{S_l}{\sum_{m=1}^{L-1} S_m}. \tag{3}$$

The probability thus defined tends to choose a unit which is most sensitive in reducing the error. Once a candidate unit is chosen, a similar probability is defined to choose the connections for this unit. For the incoming connections, we have

$$\Pr(w(i, n(l)+1, l) \text{ is chosen given the unit at layer } l \text{ chosen}) = \frac{S_{i(n(l)+1)l}}{S'_l}, \tag{4}$$

where S'_l is obtained through summing the sensitivies over both its incoming and outgoing connections, and $w(i,n(l) + 1,l)$ is the incoming connection for the chosen candidate unit for $1 \le i \le n(l-1)$. For the outgoing connections, we have

$$\Pr(w(n(l)+1, j, l+1) \text{ is chosen given the unit at layer } l \text{ chosen}) = \frac{S(n(l)+1)j(l+1)}{S_l'} \tag{5}$$

where $w(n(l) + 1,j,l + 1)$ is the outgoing connection for 1

$\leq j \leq n(l+1)$, and $S_{(n(l)+1)j(l+1)}$ is its corresponding sensitivity.

One complete adding phase is done as follows. Each incoming connection of a candidate unit is initially set to zero while its outgoing connections are picked to be small random numbers. Then the sensitivity of each incoming weight is evaluated and the sensitivity of each candidate unit is obtained through eqn (2). The probability rule defined in eqn (3) is then used to select a unit. Here we assume only one unit is added each time. Specifically, if a number generated randomly from a uniform distribution on (0,1) exceeds the probability calculated from eqn (3), the candidate unit $n(l)+1$ is selected. Once a unit is chosen, the same procedure is applied to choose connections for this unit using the probability defined in eqn (5). The weights associated with the newly added unit are then adapted while the old ones are kept frozen. After that, all the weights are trained simultaneously until a new local minimum is reached.

2.2. Deleting Phase

The deleting phase is the inverse of the adding phase, i.e. connections which cause the least error increase should be removed. The deleting scheme is based on an exhaustive search:

$$\text{Set } w(i,j,l)=0 \quad \text{if} \quad E(w(ij,l)=0)$$
$$= \min_{w(k,m,n)\neq 0} E(w(k,m,n)=0), \tag{6}$$

where $1 \leq i \leq n(l-1)$, $1 \leq j \leq n(l)$ and $1 \leq l \leq L$. Here $E(w(k,m,n)=0)$ is the error obtained when the weight $w(k,m,n)$ is set to zero. That is, each weight is set to zero consecutively and the corresponding error is computed and stored. The weight which causes the smallest error increase is deleted. This approach always decides correctly which weight should be removed. The computational complexity is polynomial in W ($\sim O(W^3)$) since computing the corresponding errors for all nonzero weights costs about MW^2 multiplications, and M is at most $O\left(\frac{W}{\epsilon}\right)$, which is chosen according to the theoretical result (Baum & Haussler, 1989). Here W is the total number of independently modifiable weights of the network and M is the total number of training samples. More recent sophisticated deleting rules can be found in (Hassibi & Stork, 1992).

In our experience, the most computational cost actually lies in the adding phase. If a network structure is built up carefully, computation time spent on exhaustive search in the deleting phase can be minimized.

2.3. The Algorithm

The algorithm uses a training set to modify the weight values and a validation set to obtain an estimate for the generalization error. Specifically, the algorithm runs as follows.

1. Train an initial network until the change of error $\sum_{i,j,l}\left(\frac{\partial E}{\partial w(i,j,l)}\right)^2$ is smaller than a specified small quantity δ where a local minimum is reached. Then compute the validation error ϵ_{val} at this local minimum, and go to (2).
2. Add resources by completing the adding phase. Then train the network until another local minimum is reached. At this local minimum a new validation error ϵ_{val}' is obtained.
3. Choose a small positive quantity δ' to evaluate the change between the two validation errors at the two consecutive local minima.
4. If $\epsilon_{val} - \epsilon_{val}' > \delta'$, i.e. the validation error decreases through adding resources, the added resources have improved learning. Then go back to (2). Otherwise, repeat the adding process one more time to make sure that the addition can not improve generalization any further. If this causes the validation error to decrease again, go to (2). Otherwise, got to (3).
5. Apply the deleting phase repeatedly until ϵ_{val} goes up. Then go back to the adding phase one more time until a new local minimum is reached and ϵ_{val}' is obtained. If $\epsilon_{val} - \epsilon_{val}' > \delta'$, set $\epsilon_{val}' \rightarrow \epsilon_{val}'$, and go to (2). Otherwise, stop.

During the entire procedure always keep a record of the weights which have yielded the smallest ϵ_{val}.

3. SIMULATIONS: HANDWRITTEN DIGIT RECOGNITION

The algorithm was first tested on handwritten digit recognition. Our goals include: (1) to test whether the algorithm is capable of finding the smallest network possible for the problem; (2) to compare the generalization performance of small networks with that of large ones. In what follows, we will show that the algorithm is indeed capable of finding small networks generalizing well. In the meantime, the comparison between small and large networks reveals an unexpected good generalization performance of large networks.

The training set contains 450 10×10 binary (1,0) images of handwritten digits: 3s, 6s and 8s obtained from the Post Office zipcode data. The validation set has 150 samples. The test set contains 440 samples. All the simulations were done twice using different initial conditions. Different network sizes are obtained from different parameter δ', which controls the variation on validation error and thus controls the size of the resulting network. In general, one would end up with a smaller network if one could wait long enough for the validation error curve to flatten out by using a very small δ'. For the same δ' chosen, networks with different initial conditions result in the similar network structure consistently.

TABLE 1
Experimental results of ADDEL for the handwritten digit recognition with and without rejection

Nets	E(train)	E(test)/E(reject)	E(test)/E(reject)
101-2-0-3*[1]	2.22	7.27/0	4.77/7.00
101-5-0-3*[2]	1.20	7.50/0	5.00/8.41
101-2-2-3*[3]	2.00	8.86/0	5.22/17.0

"*1", "*2" and "*3": the networks obtained using ADDEL algorithm with $W = 180$, 240 and 82 respectively, where W is the total number of weights of the network.
E(train), E(test) and E(reject): training, testing and rejection error rates (%) respectively.

3.1. Finding Small Networks

We first examined the capability of the algorithm to find the smallest network needed by the problem. Choosing different δ's, we obtained networks with different number of weights (Table 1) where the smallest δ results in the smallest network (*3), and the largest δ leads to the largest network (*2).

The results show that the network with the smallest number of units is the network with two hidden units and sparse connections from the input layer to the hidden layer (*1 in Table 1). To verify that this is indeed the minimum network required by the problem, multi-dimensional discriminant analysis (Duda & Hart, 1973) has been carried out to find the two dominant eigenvectors of the scatter matrices of the digits. The 450 digits 3, 6 and 8s are then projected onto the two-dimensional space spanned by these two eigenvectors. Figure 1 shows that the three types of digits are well-clustered in this reduced two-dimensional space where two hidden units will be sufficient to perform the classfication. Then in the original 100-dimensional space, a two-layer net with two hidden units will most likely be the minimum network needed for the problem. This verifies our experimental results.

3.2. Performance Comparison with Large Networks

To compare the the performance of the resulting small networks with that of large ones, we also trained networks larger than necessary on the same data set using back-error-propagation (BEP). The results are given in Table 2.

Compared to Table 1, these results show that the large networks have comparable test errors to the small (two-layer) networks obtained through ADDEL when the rejection rate is zero. When rejections are allowed to achieve the same test error, around 5%, the small (two-layer) networks usually have smaller rejection rates than the larger ones, except for the (101-10-2-3) network. If the network is too small (too few connections), for instance, the (101-2-2-3) net, its performance on the test samples may deteriorate, possibly due to insufficient resources.

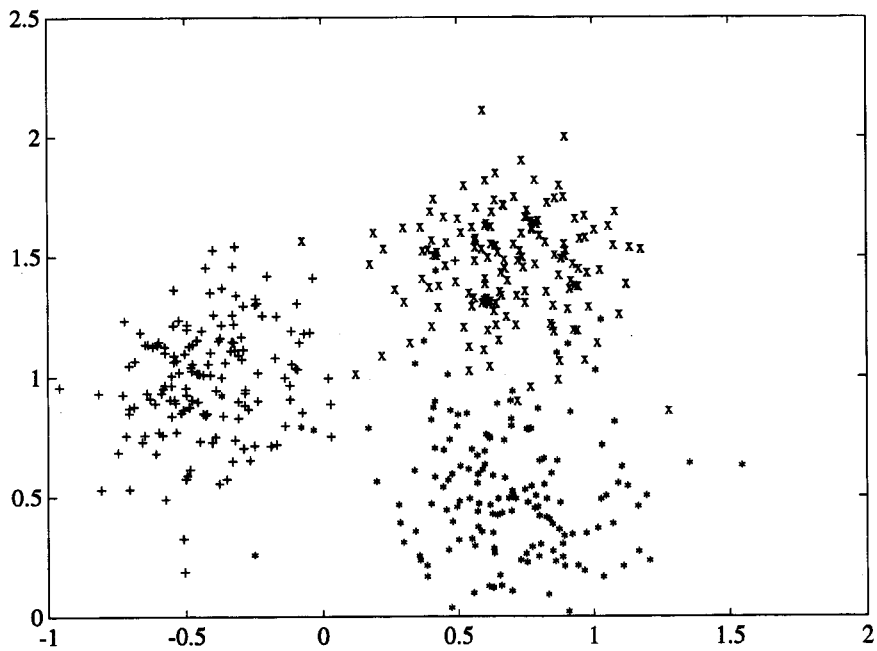

FIGURE 1. Digits projected onto the two-dimensional space spanned by the two eigenvectors. "*", " + " and "x": 3s, 6s and 8s, respectively.

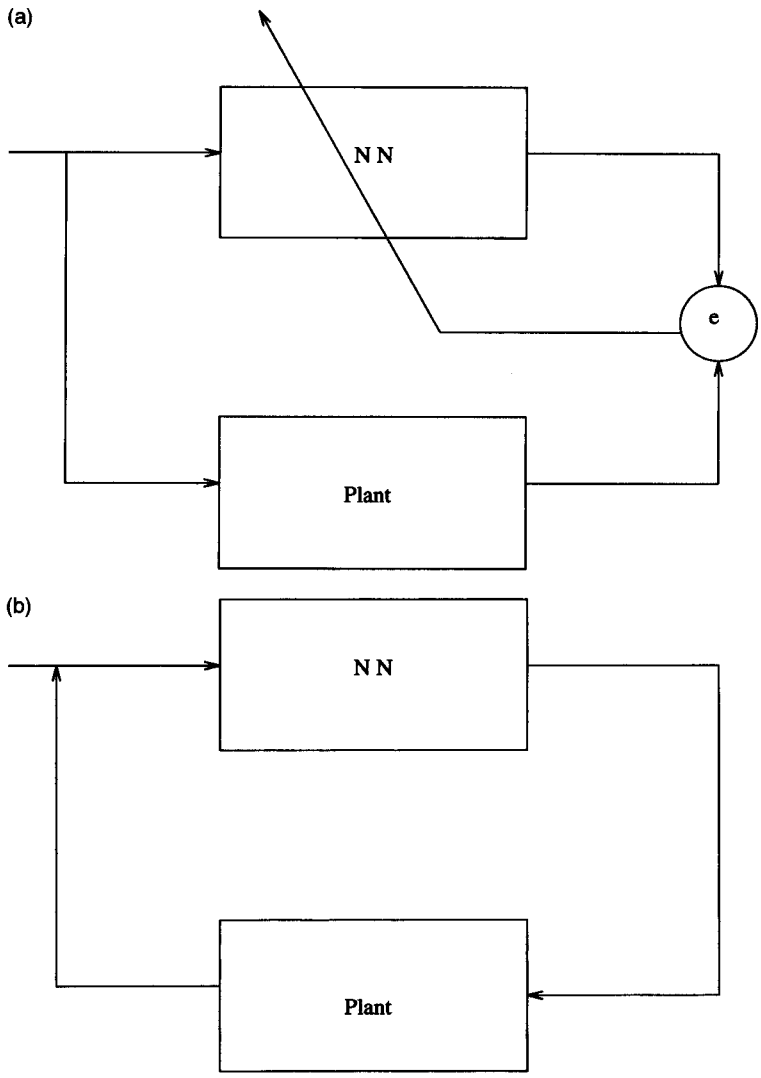

FIGURE 2. (a) Off-line training system, and (b) on-line control system. "NN" and "Plant" stand for the neural network and two-link manipulator respectively. "e": error between the outputs of the neural network and the plant.

More comparisons are made by testing both large and small networks on two types of noisy input patterns. First, the noisy patterns are generated, for which the 1s in each image are changed to 0s with a probability of 0.05. Second, patterns with additive white noise are produced by adding a number generated from the uniform distribution in [0, 0.5] to each bit. The results shown in Table 3 demonstrate that small networks with two layers are more robust to the (1–0) noise, while large networks are less sensitive to the additive white noise.

One important feature exhibited by all these results is that networks larger than necessary, in terms of the number of weights, can generalize surprisingly well. Similar results have also been reported in Lee (1991), Martin & Pittman (1991) and Geman, Bienenstock and Doursat (1992). Lots of effort has been made to provide answers to this paradoxical generalization of large networks including various measures on the effective complexity of networks (Amari & Murata, 1993; Moody, 1991). In what follows, we will discuss two aspects of

TABLE 2
Simulation results for the handwritten digit recognition with and without rejection using large networks trained with backpropagation

Nets	E(train)	E(test)/E(reject)	E(test)/E(reject)
101-10-2-3	2.00	7.24/0	5.00/5.91
101-20-0-3	0.89	7.72/0	5.00/10.7
101-40-0-3	1.05	7.72/0	5.00/8.64

TABLE 3
Test results on the noisy handwritten digits

Nets	E(1-0)/E(1-0 reject)	E(w-n)/E(w-n-reject)
101-2-0-3*[1]	8.18/0, 5.00/13.4	16.6/0, 8.18/51.6
101-5-0-3*[2]	8.41/0, 5.23/12.3	18.6/0, 5.00/44.8
101-2-2-3*[3]	11.1/0, 5.23/26.8	20.7/0, 10.0/50.0
101-10-2-3*	9.55/0, 5.00/9.32	13.0/0, 5.23/30.0
101-20-0-3*	11.4/0, 5.23/19.5	12.7/0, 5.00/25.7
101-40-0-3*	10.0/0, 5.23/13.9	8.41/0, 5.00/9.32

E(1-0) and E(1-0 reject): the test error and rejection rates on digits with the 1-0 noise.
E(w-n) and E(w-n-reject): the test error and rejection rates on digits with the white noise.
The rest of the notation follows that in Tables 1 and 2.

the problem which have been explored very little, namely, how the generalization performance of large networks varies with: (1) different types of problems, and (2) the dynamics of gradient decent algorithm. We will mainly discuss the former aspect in the next section, and briefly mention the work related to the second aspect of the problem.

4. PROBLEMS FOR WHICH LARGE NETWORKS ARE HARD TO GENERALIZE

We expect that the proper size of a network is related to the problem. Intuitively, "difficult" problems require a precise match between the network complexity and the problem complexity in order to achieve good generalization. Roughly speaking, for binary mappings, the difficulty of a problem can be characterized by how well samples belonging to different classes are clustered. In what follows, we will show examples of "difficult" problems for which small networks are more likely to generalize well than large ones. The results can suggest a precise definition of the difficulty of a problem as well as its effect on required network complexity. This topic will be investigated further in future research.

4.1. Learning a "Critical" Target Perceptron

To investigate whether there is a problem for which large networks do not generalize well, a simple example is chosen in which networks of different sizes are trained to learn a target network which is a single-neuron (15-1) network with random weights. The training samples generated independently from an uniform distribution in $[-3,3]$ are fed through the target perceptron to obtain their labels. Two networks (15-1 and 15-20-1) were trained with BEP using 600 training samples, and then tested on another 1000 randomly drawn samples from the same distribution. Their training and test error rates are all below 2.5 and 4.2%, respectively. However, when both networks are tested on 1000 samples that were selected to lie very close to the original decision boundary (i.e. samples whose outputs are in the range $[-0.1, 0.1]$ at the output of the random target network), they both make about 50% errors on the test samples! The reason for the apparent success previously is that there are only 0.2% samples in the 1600 samples used which are close to the boundary. That is, the randomly drawn samples from two classes are well separated. So the resulting small network is actually tilted away from the original decision boundary while the large one is wiggling around it.

TABLE 4
Simulation results for learning the critical target perceptron

Nets	E(train) (%)	E(test) (%)
15-1*	8.70	9.80
15-2-1	6.89	9.40
15-10-1	5.50	53.0

"*": trained using ADDEL which resulted in 14 connections; and the other two networks were trained using BEP along with the validation set.
E(train) and E(test): the training and test error rates.

To investigate a case for which an under-trained large network does not generalize well, we use training and validation sets of the same sizes as before, but the samples are only those close to the boundary (within the $[-0.1, 0.1]$ range) defined by the target perceptron. Similar training for networks with different sizes is carried out; and the results are given in Table 4. The networks that are trained all start with very small random initial weights in the range $[-a,a]$, where $a < 10^{-5}$. Being consistent with the perceptron convergence theorem (Duda & Hart, 1973), ADDEL results in a single neuron with 14 connections which generalizes well. The 15-2-1 network trained with BEP also converges to a network with equivalently the same number of weights as the target perceptron (some of the weights in the resulting networks are almost zero), although the loading of the training samples takes quite long. The 15-10-1 network, however, learns the training samples but fails completely to learn the mapping.

4.2. Handwritten Digit Recognition Revisited

Similar arguments can be provided for handwritten digit recognition to explain why large networks can

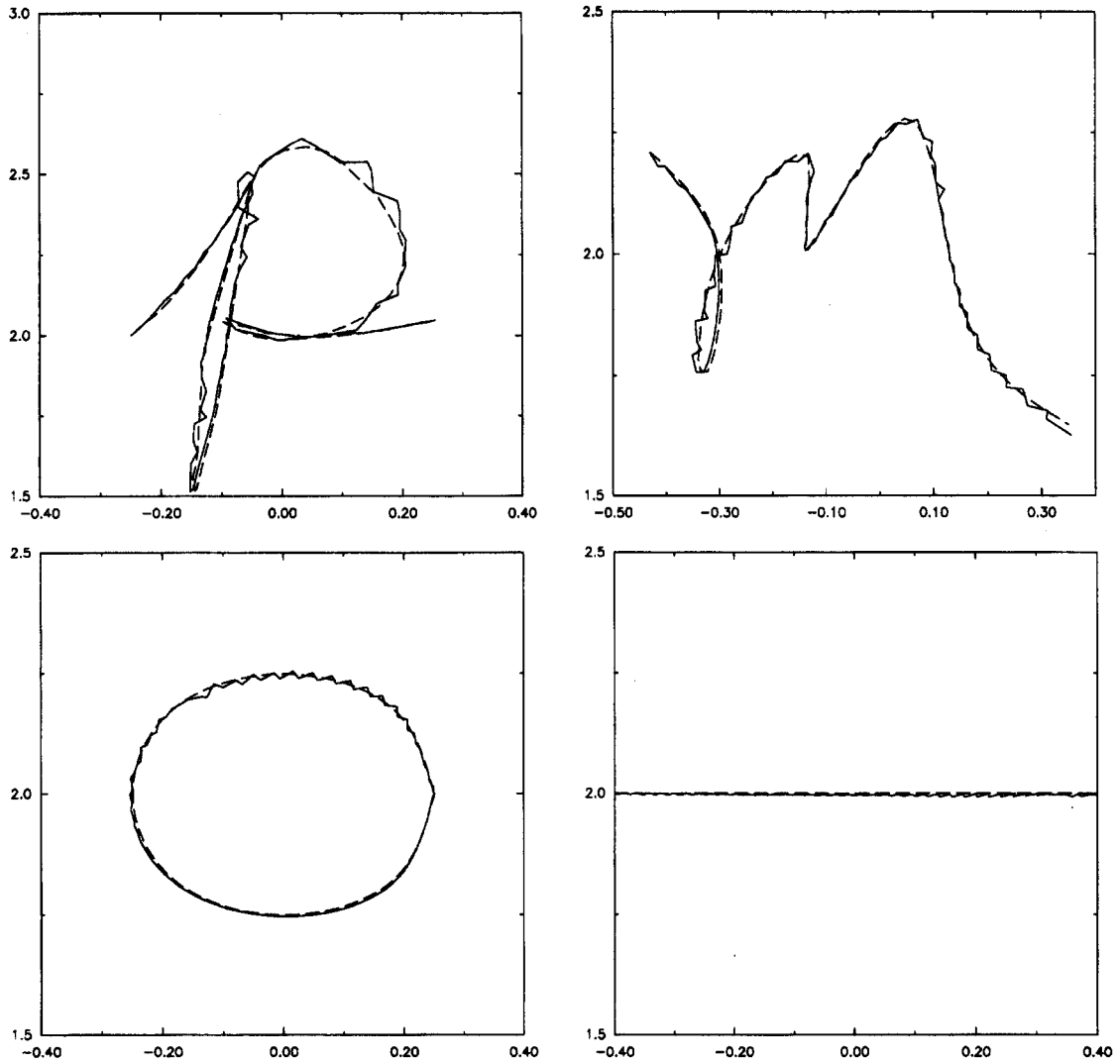

FIGURE 3. Characters drawn by a network trained with the algorithm ADDEL. The dashed and solid curves are the teacher trajectories and the actual trajectories given by the networks, respectively.

also generalize well. Consider Figure 1 again. Since even in the reduced two-dimensional space, the samples belonging to different classes form well-separated clusters, it is natual to imagine that in the original 100-dimensional space, the samples will be even better separated. Therefore, the well-seperated clusters formed by the digits make this classification problem "easy", which is similar to the situation in learning the target perceptron that generates well-clustered samples.

It should be noted that the relatively small training set size also makes the problem somewhat easier. This is because a large training set may reveal a more complicated data structure so that the corresponding classification error is more sensitive to the size of the network.

4.3. Learning an Analog Mapping

In general, analog mappings are more difficult to learn than binary mappings, since every sample point needs to be learned to a certain degree of accuracy for its analog value, instead of only the signs around a threshold as in the case of binary mappings. In what follows, we show experimentally that the generalization performance is sensitive to the size of a network in learning an analog mapping.

The problem considered is the off-line training of a network shown in Figure 2 to approximate an inverse plant to control a two-link manipulator to draw characters. The dynamics of the two-link manipulator are described by the Newton-Euler equation (Craig, 1986). The task is to train a network to draw characters inside a window. The training set contains 100 desired inputs $\underline{\theta}(\underline{\theta}' = (\theta_1, \theta_2)), \dot{\underline{\theta}}, \ddot{\underline{\theta}}$, which are the angular position, velocity and acceleration vectors, and desired outputs $\underline{\tau}(\underline{\tau}' = (\tau_1, \tau_2))$, which are the torques to be applied to the two-links. These input and output samples are drawn from a single trajectory for drawing the letter *P*. For more details on the mapping see Craig (1986).

TABLE 5
Simulation and test results for drawing characters, trained and untrained

Character drawn	Hidden units	Number of failures	MSE
P	3*	0	0.005
(training samples were drawn from this character)	10	0	0.046
	20	9	**
M	3*	0	0.005
	10	0	0.009
	20	7	**
Circle	3*	0	0.004
	10	1	0.011
	20	7	**
Line	3*	0	0.004
	10	1	0.013
	20	6	**

"*": networks trained with ADDEL.
"10" "20": networks with 10 and 20 hidden units trained with BEP.
"Number of Failures": number of networks out of 10 networks which draw the character unsuccessfully.
"**": the MSE can not be evaluated due to the overflow of the numbers in failures of drawing the characters.
MSE $= \sum_{i}^{K} \|\underline{\theta_{ai}} - \underline{\theta_i}\|^2$, where $\underline{\theta_{ai}}$ and $\underline{\theta_i}$ are the i-th test sample for the angular positions from the actual and ideal trajectories respectively.

The off-line training process is equivalent to finding a network which approximates the analog mapping $f: R^6 \rightarrow R^2$. A more restricted version of the algorithm is used, which only adds and deletes units at a two-layer structure, resulting in 8 networks with 3 hidden units out of 10 simulations starting with different random initial conditions. Large networks with 10 and 20 hidden units were also trained 10 times each using BEP. The resulting networks are then connected in a feedback fashion, as shown in Figure 2b, with the simulated two-link manipulator to test the generalization.

The expected generalization is twofold: (1) how well the resulting networks generalize from the discrete training points to the whole trajectory for drawing the letter P? (2) can an unseen character other than P be drawn successfully? That is, the resulting networks are expected to generalize from discrete samples on a specific trajectory to the other trajectories in the input space. The normed difference between the angular positions yielded from the actual trajectory and the desired trajectory is used to evaluate the generalization quantitatively. The resulting networks are tested on the trained P as well as the unseen character M, a circle and a straight line. The results given in Table 5 demonstrate that small networks generalize consistently better than large ones. However, the latter can learn each individual training point well, but fail even to follow the whole trajectory for the trained letter P. Some unseen characters successfully drawn by the small networks are given in Figure 3. Further testing is done on the translated, rotated and scaled versions of the Ps. The results are given in Table 6. In this case, the small networks out perform the large ones even more convincingly.

4.4. Initial Complexity

In our recent work (Atiya & Ji, 1997), we investigated another aspect of the problem, namely, the effect of initial conditions on the generalization of resulting networks.

Intuitively, the strong influence of initial conditions on generalization performance of resulting networks can be attributed to the intrinsic properties of gradient decent methods and the under-training of a large network. That is when the step size used is sufficiently small, gradient decent methods search for a solution only within a neighborhood of initial conditions. Therefore, the information on network complexity determined by the initial distribution of weights will be carried over to the complexity of resulting networks. The extreme case is that the initial network is linear due to the small random initial weights. Then the nonlinearity grows during learning. In short, the combination of the initial condition and the gradient decent type of algorithms actually limit the search space that the algorithm may explore, and therefore eliminate the number of networks possible determined by the architecture itself. This contributes another factor to the good generalization

TABLE 6
Test results for the shifted, rotated and scaled *P*

dx	dy	θ (degree)	Scale	Hidden units	Number of failure	MSE
0	0	0	0.9	3*	1	0.009
				10	5	0.046
				20	10	**
0	0	10.0	1.0	3*	0	0.007
				10	5	0.215
				20	10	**
− 0.3	− 0.1	0	1.0	3*	1	0.007
				10	3	0.073
				20	10	**

performance of large networks as in the handwritten digit recognition. It remains an open problem to characterize the problems for which large networks that generalize well can be easily found versus problems for which small networks (such as the ones created by ADDEL) can be obtained through effective training methods.

5. CONCLUSION

The algorithm presented in this paper consistently finds small networks that give good generalization. The examples we consider indicate that, in learning both analog and binary mappings, small networks are more likely to achieve good generalization only if a problem is "difficult".

In terms of learning time, the under-trained large networks usually learn faster than the small ones. For instance, the average runing time for ADDEL for the 3-6-8 problem is about twice that for the large networks with 20 hidden units, if it is required to obtain a small network with about 150 connections.

In terms of choice of an algorithm, it is certainly best to incorporate clever designs (Sackinger, Boser, Bromley, LeCun & Jackel, 1992) into a network if some a priori knowledge is available, then to use a data-driven approach like ADDEL to find the complete network architecture. However, even if no a priori knowledge is available, the network addition–deletion algorithm can be reliably used to find a structure which will yield good generalization.

REFERENCES

Amari, S., & Murata, N. (1993). Statistical theory of learning curves under entropic loss criterion. *Neural Computation, 5,* 140–153.

Atiya, A. & Ji, C. (1997). How do initial conditions affect generalization performance of large Networks? *IEEE transactions on Neural Networks, 8*(2), 448–451.

Barron, A. (1991). Approximation and estimation bounds for artificial neural networks. *Proc. of The 4th Workshop on Computational Learning Theory* (pp. 243–249).

Baum, E., & Haussler, D. (1989). What size net gives valid generalization? *Neural Computation, 1,* 151–160.

Craig, J. J. (1986). *Introduction to robotics: mechanics and control.* New York: Addison-Wesley.

Denker, J. S., LeCun, Y. & Solla, S. A. (1989). Optimal Brain Damage. In D. Touretzky (Ed.), *Advances in neural information processing systems,* Vol. 2, (pp. 598–605). San Mateo: Morgan Kaufmann.

Duda, R. & Hart, P. (1973). *Pattern classification and scene analysis.* New York: John Wiley and Sons.

Fahlman, A.E. & Lebiere, C. (1989). The cascade-correlation learning architecture. In D. Touretzky (Ed.), *Advances in neural information processing systems,* Vol. 2, (pp. 524–532). San Mateo: Morgan Kaufmann.

Frean, M. (1990). The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation, 2,* 198–209.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4,* 1–58.

Hassibi, B. & Stork, D.G. (1992). Second order derivatives for network pruning: optimal brain surgen. In D. Touretzky (Ed.), *Advanes in neural information processing systems,* Vol. 5, (pp. 164–171). San Mateo: Morgan Kaufmann.

Ji, C., Snapp, R., & Psaltis, D. (1990). Generalizing smoothness constraints from discrete samples. *Neural Computation, 2,* 190–199.

Lee, Y. (1991). Handwritten digit recognition using k nearest neighbour radial-basis function, and backpropagation. *Neural Networks, 3,* 440–449.

Martin, G.L., & Pittman, J.A. (1991). Recognizing hand-printed letters and digits using back propagation learning. *Neural Computation, 3,* 258–267.

Moody, J. (1991). The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In D. Touretzky (Ed.), *Advances in neural information processing systems,* Vol. 4, (pp. 847–854). San Mateo: Morgan Kaufmann.

Nadel, J. (1989). Study of a growth algorithm for neural networks. *International Journal of Neural Systems, 1,* 55–59.

Nowlan, S.J., & Hinton, G.E. (1992). Simplifying neural networks by soft weight sharing. *Neural Computation, 4,* 473–493.

Sackinger, E., Boser, B. E., Bromley, J., Lecun, Y., & Jackel, L. D. (1992). Application of the anna neural network chip to high-speed character-recognition. *IEEE Transactions on Neural Networks, 3,* 498–505.

Weigend, A., Rumelhart, D. E. & Huberman, B. A. (1990). Generalization by weight elimination with application to forecasting. In D. Touretzky (Ed.), *Advances in neural information processing systems,* Vol. 3, (pp. 875–882). San Mateo: Morgan Kaufmann.